15-451/651 Algorithms, Fall 2017

Homework #2 **Due: September 19–22, 2017**

(25 pts) 1. **(Balls in Bins)** There are n balls and an infinite number of bins. A bin can have 0 or more balls in it. A move consists taking all the balls of some bin and putting them into distinct bins. The cost of a move is the number of balls moved.

Define the potential of a state of this system as the sum of the potentials of all the bins. The potential of a bin with k balls in it is:

$$\Phi(k) = \max(0, k - z)$$

Where for convenience $z = \lfloor \sqrt{n} \rfloor$.

Prove that the amortized cost of a move is at most $2z$.

Solution: Consider one move in which k is the number of balls in the bin vacated. Let ac be the amortized cost of the move. Let X be the set of bins whose count increases where the final number of balls is $> z$. (These are precisely the bins causing the potential to increase.) Breaking the analysis into two cases we get:

$$ac = \begin{cases} k + |X| & \text{if } k \leq z \\ k - (k - z) + |X| & \text{otherwise} \end{cases}$$

Clearly $|X| \leq k$, so in the first case $ac \leq 2k \leq 2z$.

We also know that each bin of X has at least $z + 1$ balls in it at the end. So we know that $|X| * (z + 1) \leq n$. So $|X| \leq z$. (If $|X| \geq z + 1$ then $|X| * (z + 1) > n$, a contradiction.) So in the 2nd case we have:

$$ac = k - (k - z) + |X| \leq k - (k - z) + z = 2z$$

which finishes the proof. ■

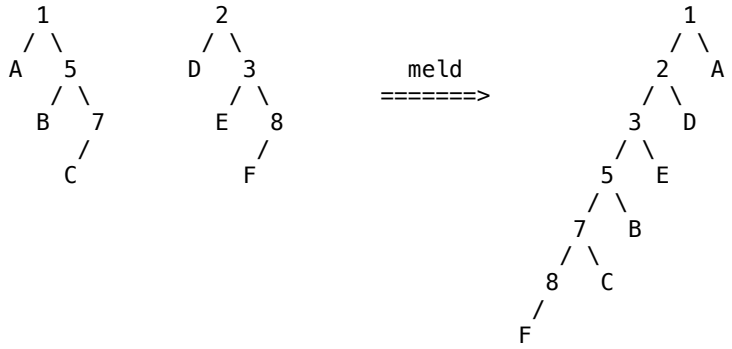
(25 pts) 2. (**Cheap Heaps)** A heap is a data structure to store a multi-set of keys which supports the following operations:

`meld(A,B):`	Return the heap representing the union of the contents of the two heaps `A` and `B`.
`findmin(A):`	Return the minimum key of heap `A`.
`deletemin(A):`	Delete the minimum key from `A` and return the resulting heap.
`insert(A,x):`	Insert the key `x` into the heap `A` and return the new heap.

A *cheap heap* implements this interface as follows. The cheap heap is a binary tree, where each node stores one key. The tree is heap ordered, meaning that the key in a node is at most that of its children.

`findmin(A)` simply returns the key in the root of `A`. All the other operations are implemented using `meld`. I.e. `deletemin(A)` deletes the root of `A` and melds its two children together. `insert(A,x)` creates a new tree with one node containing `x` and melds it with `A`.

To implement `meld(A,B)` we walk down the right paths from the roots of `A` and `B` and merge those paths together to form a new right path of non-decreasing keys. Now for each node along that path (except the last one) we swap the left and right children. The following figure shows the `meld`ing of two trees. (The subtrees labeled `A`, `B`, etc. are arbitrary and do not change. They could be empty.)



WLOG define the running time of `meld` to be the number of nodes on the melded paths. (This would be six in the example above.)

(a) Define a potential function of a cheap heap. The potential of a collection of cheap heaps is just the sum of each individual tree's potential. Your potential will be used to solve the remaining parts of this problem.

Hint: Define the *size* of a node in the cheap heap as the number of nodes in the subtree rooted there (as we did with splay trees). This allows you to classify edges of the tree to be *light* or *heavy* as follows. If the node at the top of the edge has size A and the one at the bottom has size B, and $A \geq 2B$ then the edge is light. Otherwise it is heavy. You can also classify an edge to be *right* if node B is a right child of A and *left* otherwise. The potential function will be the number of edges of a certain type in this classification system.

Solution: Define the potential to be the number of heavy right edges.

(b) Say that two trees are `melded` to produce a tree of size n. Using your potential function prove that the amortized cost of this operation is at most $1 + 4\log n$. (Even better is $1 + 3\log n$.)

Solution: Partition the edges of the right path down from A into the sets A_ℓ and A_h, the light and heavy ones respectively. Define B_ℓ and B_h similarly. The cost of the meld is $|A_\ell| + |A_h| + |B_\ell| + |B_h| + 1$. The potential of these two paths before the meld is $|A_h| + |B_h|$.

Let's consider the melded path M after the flip (all left edges). Let M_ℓ be the light edges on this path. The potential of the the edges that replace the A and B paths is at most $|M_\ell|$, because any heavy edge that sticks off the right of the M path must have a sibling light left edge in M_ℓ. Therefore the increase in potential is at most

$$|M_\ell| - |A_h| - |B_h|.$$

So the amortized cost of the operation is at most:

$$(|A_\ell| + |A_h| + |B_\ell| + |B_h| + 1) + (|M_\ell| - |A_h| - |B_h|) \leq 1 + 3\log n$$

Where the last inequality follows from that fact that there can be at most $\log n$ light edges on any path. ■

(c) Suppose initially there are no cheap heaps, and a sequence of m of the above listed operations are done, and that no heap ever exceeds size n. Prove that the total cost of all the operations done is $O(m \log n)$.

Solution: Recall from lecture that:

$$\text{total cost} = \sum_i c_i = (\sum_i ac_i) + \Phi_{\text{initial}} - \Phi_{\text{final}}$$

The initial potential is zero, and the final potential is at least 0. Therefore:

$$\text{total cost} \leq \sum_i ac_i \leq m(1 + 3\log n) = O(m \log n)$$

By the way, below is the complete Ocaml code to implement cheap heaps.

```
type 'a tree = Empty | Node of 'a tree * 'a * 'a tree

let rec meld a b =
  match (a,b) with
  | (Empty, b) -> b
  | (a, Empty) -> a
  | (Node(al, ak, ar), Node(bl, bk, br)) ->
```

```
      if (ak <= bk) then Node((meld ar b), ak, al)
      else Node((meld a br), bk, bl)

let insert a k =
  meld a (Node(Empty, k, Empty))

let findmin a =
  match a with Empty -> failwith "Findmin on empty heap"
  | Node(al, ak, ar) -> ak

let deletemin a =
  match a with Empty -> failwith "Deletemin on empty heap"
  | Node(al, ak, ar) -> meld al ar
```

(25 pts) 3. (**Trade-offs**) Suppose we want a data structure that represents a sequence of numbers $x_1, x_2, \ldots, x_n$ (initially 0), with the following operations:

Increment(i, v): $x_i \leftarrow x_i + v$

SumToLeft(j): return $\sum_{1 \le i \le j} x_i$

Let $f(n)$ be an asymptotic bound on the cost of the Increment() operation and let $g(n)$ be an asymptotic bound on the cost of the SumToLeft() operation.

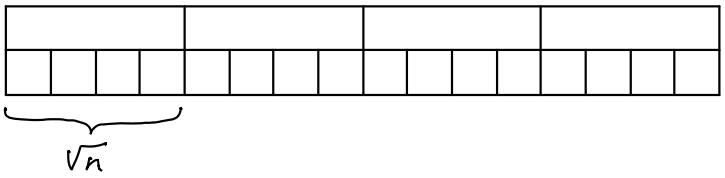
(a) Show how to obtain $(f(n), g(n)) = (O(1), O(\sqrt{n}))$.

(b) Show how to obtain $(f(n), g(n)) = (O(1), O(n^{1/3}))$.

(c) Show how to obtain $(f(n), g(n)) = (O(\sqrt{n}), O(1))$.

(d) Show how to obtain $(f(n), g(n)) = (O(n^{1/3}), O(1))$.

Solution: All of these solutions are generalizations of Fenwick Trees. For part (a) there are two levels, the bottom level has n blocks of size 1. The level above it has $\sqrt{n}$ blocks of size $\sqrt{n}$. Here is a picture:



To do Increment(i, v) we add v to the lower level block at position i and the upper level block above it. This is $O(1)$. to do SumToLeft(j) we begin by adding in up to $\sqrt{n} - 1$ blocks on the lower level, then at most $\sqrt{n}$ blocks above it. This is $O(\sqrt{n})$.

In part (b) we simply use three levels. The lower has blocks of size 1, the next level has blocks of size $n^{1/3}$ and the third level has blocks of size $n^{2/3}$. The bounds hold.

For parts (c) and (d) we use the trick described in the lecture 5 at the bottom of page 6:

Another observation is that the theorem about these paths having a unique intersection when $a < b$ means that we can swap the role of `UP-PATHS` and `DN-PATHS`. So we could use `DN-PATH`s for the Increments and `UP-PATH`s for the sums. In this case the sum would be for all items from i to n.

Using this idea we can swap the running times of the two operations, which gives solutions to parts (c) and (d).